Reproducible Backtesting via Contract Hashing

Author · NebulaBunny Research Lab

Abstract

There are reproducibility problems in financial backtesting: Unrecorded parameter drift, silent data updates and mismatch with the actual implementation.

I propose a deterministic, contract-driven architecture that can achieve byte-level reproducibility under controlled conditions.

Each experiment is defined by a encrypted fingerprint – **spec_hash**, which binds strategy logic, fee and slippage models, data fingerprint and random seed together.

Each run will generate an independent **FREEZE** capsule containing results, logs and verification values.

No remote services are required for validation: replay the capsule in the declared environment to reproduce the same digest.

The framework introduces the BPS between the simulated transaction and the actual execution to carry out the quantitative deviation audit, which connects the backtesting and the reality verification.

1 Introduction

Reproducibility is still a long-standing problem in the field of quantitative finance.

Subtle library or seed differences may quietly change performance and cause butterfly effect.

Our framework regards each backtest as a contract between code, data and environment; The generated **spec hash** becomes an immutable unique identifier! It makes independent verification possible and does not leak logic at the same time.

2 Methodology

2.1 Contract and Hash Binding

Each experiment is represented by

$$\Sigma = (E, C, F, S, R)$$

where

E=engine specification, C=contract (fee, slippage, risk), F=data fingerprint, S=source hash, R=random seed.

$$spec\_hash = H(E + C + F + S + R)$$

Any change of these parameters will generate a new hash value, which is encrypted to lock the experimental range.

2.2 The FREEZE Capsule

A **FREEZE** archive contains
JSON files of results and reports,
Dependency list,
The entered verification value,
Fingerprints: code_git_hash, data_version, spec_hash, random_seed, env_fingerprint
Log and KPI summary at runtime.

verification:

1. open the capsule packaging.
2. run the replay utility with the declared dependencies.
3. compare the SHA256 of report.json to confirm the repeatability.

2.3 Deterministic Execution (Practical Version)

he determinacy is achieved through fixed dependent version, explicit random seed and single threaded computing, in order to minimize the impact of uncertain scheduling.

All timestamps are UTC standardized, and the text encoding is fixed to UTF-8.

If it is feasible, the standard BLAS/LAPACK library is used to improve the consistency of values, but it can't guarantee the absolute binary equivalence in all environments.

In the preliminary cross platform test (Linux and Windows are under the same dependency), the report with the same byte is generated; The ongoing verification will also expand the sample set and record the exception.

---

3 Drift Audit and Live Validation

The shadow matching layer replays real-time transactions with the same logic.
Income per transaction:

$$drift_{bps} = \frac{P_{shadow} - P_{real}}{P_{real}} \times 10^4$$

Among them, P_{shadow} and P_{real} are simulation price and actual price respectively.
The distribution (p50, p90, p95, p99) quantifies the consistency between the model and the market.
The historical interest rate playback is used to check the fund cost equivalence.
The results will be recorded in alignment audit.json.

---

4 Experimental Evaluation

The prototype runs under a controlled dependency and produces identical hashes across machines.

The drift audit of BTC/ETH/SOL shows that, under normal circumstances, the median drift is less than 1 basis point.

In the expansion of the boundary distribution caused by extreme events (such as the volatility in March 2024), the robustness of the deterministic matching kernel is also proved.

---

5 Discussion and Future Work

The contract hash framework transforms the backtracking test into a verifiable scientific experiment.

The future directions include formal determinism proof, multi-party audit network, encrypted signature of **FREEZE** package, and the combination of complexity penalty and stability measurement out of sample to limit over fitting.

Founder's Note

My goal is to constrain randomness as much as possible through design. In NebulaBunny, repeatability is the cornerstone of trust! We are a quantitative laboratory that can be reproduced, audited and mathematically proven. If the experiment can be accurately reproduced, it can also be precisely optimized.

Imagine your researchers working under this framework: when adjusting the cost or capital parameters within the pre declared locking range, the real-time drift will only tighten a few basis points, and generate a new spec hash at the same time, and the

FREEZE capsule will automatically update to verify. Quantitative research, risk management and regulation can now share a verifiable foundation. The freedom of evolution still belongs to you. We just make every step have mathematical traceability.

References

1. Sandve G.K. et al. (2013) *Ten Simple Rules for Reproducible Computational Research.* PLoS Comp Biol.
2. Borne K.D. (2018) *Data Science Reproducibility and Provenance.*
3. NebulaBunny Lab (2025) *Reproducible Backtesting via Contract Hashing.* Working Paper.